

BENCHMARK OF SOME NONSMOOTH OPTIMIZATION SOLVERS FOR COMPUTING NONCONVEX PROXIMAL POINTS

WARREN HARE* AND CLAUDIA SAGASTIZÁBAL†

Abstract: The major focus of this work is to compare several methods for computing the proximal point of a nonconvex function via numerical testing. To do this, we introduce two techniques for randomly generating challenging nonconvex test functions, as well as two very specific test functions which should be of future interest to Nonconvex Optimization Benchmarking. We then compare the effectiveness of seven algorithms (“CPROX,” “N1CV2,” “N2FC1,” “PBUN,” “PVAR,” “PNEW,” and “RGS”) in computing the proximal points of such test functions. We also examine two versions of the CPROX code to investigate (numerically) if the removal of a “unique proximal point assurance” subroutine allows for improvement in performance when the proximal point is not unique.

Key words: *nonconvex optimization, proximal point, algorithm benchmarking*

Mathematics Subject Classification: 90C26, 90C30, 68W40

1 Introduction

Benchmarking is an important tool in numerical optimization, not only to evaluate different solvers, but also to uncover solver deficiencies and to ensure the quality of optimization software. In particular, many researchers in the area of continuous optimization have devoted considerable work to collecting suitable test problems, and comparing different solvers performance both in Linear and Nonlinear Programming; we refer to <http://plato.asu.edu/bench.html> and <http://www-neos.mcs.anl.gov/> for an exhaustive list of different codes and related works.

Nonsmooth Optimization (NSO), by contrast, badly lacks the test problems and performance testing of specialized methods. Although the state of the art for convex nonsmooth optimization is quite advanced and reliable software is in use (see [2] and references therein), very few works in the literature include comparisons with some other method. In addition to the low availability of free NSO solvers, a possible reason is the lack of a significant library of NSO test functions.

The situation is even worse in Nonconvex Nonsmooth Optimization: there are not many test problems, and there are not many specially designed solvers either. Prior to the random gradient sampling algorithm introduced in [3], NSO methods consisted essentially in

*Both authors would like to thank Michael Overton for his assistance in using the RGS code.

†Research supported by CNPq Grant No. 383066/2004-2

“fixing” some convex bundle algorithm to adapt it to the nonconvex setting, [12], [14], [18]. The recent work [8] made a contribution in the area, by giving a new insight on how to pass from convex to nonconvex bundle methods. More precisely, in [8] the problem of computing a proximal point of a nonconvex nonsmooth function (cf. (1) below) is addressed. A full nonconvex nonsmooth algorithm based on the building blocks developed in [8] is very promising, because the more robust and efficient variants of convex bundle methods are of the proximal type, [10], [11].

In this work we provide a comprehensive suite of NSO functions and give a benchmark for the specific NSO problem of computing the proximal point of a nonconvex function, i.e.,

$$\text{find } p \in P_R f(x) := \operatorname{argmin}_{w \in \mathbb{R}^N} \left\{ f(w) + R \frac{1}{2} |w - x|^2 \right\}, \quad (1)$$

where $R > 0$ and $x \in \mathbb{R}^N$ are given. Our purpose is twofold. First, we assemble a sufficient set of relevant NSO functions in order to be able to evaluate numerically different approaches. Second, by comparing different NSO solvers performance on problems of type (1), we start tuning the method in [8] in order to use it as part of a future nonconvex proximal bundle method.

The main concern in presenting benchmark results is in removing some of the ambiguity in interpreting results. For example, considering only arithmetic means may give excessive weight to single test cases. Initial performance comparisons were developed for complementarity problems in [1]. In [4], the approach was expanded by the introduction of performance profiles, reporting (cumulative) distribution functions for a given performance metric. Performance profiles therefore present a descriptive measure providing a wealth of information such as solver efficiency, robustness, and probability of success in compact graphical form. For the numerical testing in this work, we give performance profiles in Sections 4, 5, and 6. All the statistical information on the results is reported in the form of tables in Appendix A.

For defining the NSO functions composing the battery, we consider two categories:

- randomly generated functions, either defined as the pointwise maximum of a finite collection of quadratic functions, or as sum of polynomial functions (operating on the absolute value of the point components); see Sections 4 and 5, respectively.
- two specific functions, the *spike* and the *waterfalls* functions; described in Section 6.

The maximum of quadratic functions family, in particular, is defined so that any of the generated problems (of the form (1)) are solved by $x^* = 0 \in \mathbb{R}^N$. This is an interesting feature, because it allows to measure the quality of solutions found by a given solver.

For all the functions, an *oracle* is defined, which for any given $x \in \mathbb{R}^N$ computes the value $f(x)$ and *one* subgradient for the function, i.e., some $g_f \in \partial f(x)$. As usual in NSO, we do not assume that there is any control over which particular subgradients are computed by the oracle. In this setting, rather than measuring effectiveness of an algorithm in terms of time, the number of oracle calls made should be considered.

The solvers we test in this benchmark are CPROX, N1CV2, N2FC1, RGS, PBUN, PVAR, and PNEW, corresponding, respectively, to [3], [8], [11], [12], [13], [14], and [15] (a compilation including the descriptions of PBUN, PVAR, and PNEW can be found in [16]).

The remainder of this paper is organized as follows. In Section 2 we provide the theoretical background for this work, as well as a brief description of each the algorithms benchmarked in this work. Sections 4, 5, and 6 contain the descriptions, performance profiles, and conclusions for our battery of test functions. Statistical tables for the tests are stored in Appendix A. We conclude with some general discussion in Section 7.

2 Theoretical Background and Solvers

Before describing the test functions and giving the numerical results, we recall some important concepts related to Variational Analysis and the proximal point mapping. We also give a succinct description of the solvers used for our benchmark.

2.1 Brief Review of Variational Analysis

For $f : \mathbb{R}^N \rightarrow \mathbb{R} \cup \{+\infty\}$ proper lower semicontinuous (lsc) function and point $\bar{x} \in \mathbb{R}^N$ where f is finite-valued, we use the Mordukhovich subdifferential denoted by $\partial f(\bar{x})$ in [20, Def. 8.3]. For such a function we use the term *regular* to refer to subdifferential regularity as defined in [20, Def. 7.25], and the following definitions:

- The function f is *prox-regular* at \bar{x} for $\bar{v} \in \partial f(\bar{x})$ if there exist $\epsilon > 0$ and $r > 0$ such that

$$f(x') \geq f(x) + \langle v, x' - x \rangle - \frac{r}{2} |x' - x|^2$$

whenever $|x' - \bar{x}| < \epsilon$ and $|x - \bar{x}| < \epsilon$, with $x' \neq x$ and $|f(x) - f(\bar{x})| < \epsilon$, and $|v - \bar{v}| < \epsilon$ with $v \in \partial f(x)$. We call a function *prox-regular* at \bar{x} if it is prox-regular at \bar{x} for all $v \in \partial f(\bar{x})$ [20, Sec. 13.F].

- The function f is *lower- \mathcal{C}^2* on an open set V if it is finite on V and at any point x in V the function f appended with a quadratic term is convex on an open neighbourhood V' of x ; see [20, Thm. 10.33, p. 450].
- The function f is *semi-convex* if, for some $\rho > 0$, $f + 1/2\rho|\cdot|^2$ is a convex function on \mathbb{R}^N .

For the definitions above, the following relations hold:

$$\begin{aligned} f \text{ convex} &\Rightarrow f \text{ semi-convex} \\ f \text{ semi-convex} &\Rightarrow f \text{ lower-}\mathcal{C}^2 \text{ on any open set } V \\ f \text{ lower-}\mathcal{C}^2 \text{ on an open set } V &\Rightarrow f \text{ prox-regular at all } x \in V \\ f \text{ prox-regular at } x &\Rightarrow f \text{ regular at } x. \end{aligned} \tag{2}$$

The first two implications are obvious, while the final two can be found in [20, Prop. 13.33] and [20, p. 610]. Additional relations to semismooth functions [17] can be found in [9].

- The *proximal point mapping* of the function f at the point x is defined by $P_R f(x)$ given in (1), where the minimizers can form a nonsingleton or empty set, [19], [20, Def. 1.22]. The *Moreau envelope* of f at x , also called proximal envelope, or Moreau-Yosida regularization, is given by the optimal value in (1), when considered a mapping of x .
- The function f is *prox-bounded* if for some prox-parameter R and prox-center x , the proximal point mapping is nonempty. In this case $P_R f$ will be nonempty for any $x \in \mathbb{R}^N$ [20, Ex. 1.24]. The *threshold of prox-boundedness*, denoted by r_{pb} , is then the infimum of the set of all R such that $P_R f$ is nonempty for some $x \in \mathbb{R}^N$.

The following result reveals the importance of implications (2), more specifically of prox-regularity, when considering proximal points.

Theorem 2.3 in [7] *Suppose f is a prox-bounded function, and the prox-parameter R is greater than the threshold of prox-boundedness. Then for any prox-center x the following holds:*

$$p \in P_R f(x) \Rightarrow 0 \in \partial f(p) + R(p - x) \tag{3}$$

Furthermore, if f is prox regular at \bar{x} for $\bar{v} \in \partial f(\bar{x})$, then there exists a neighbourhood V of $\bar{x} + \frac{1}{R}\bar{v}$ such that, whenever R is sufficiently large, the right hand side inclusion in (3) is necessary and sufficient for $P_R f(x)$ to be a singleton. As a result, $P_R f(\cdot)$ is single-valued on V .

When generating our collection of NSO functions, we make use of statement (3) to define, in addition to the test function, a prox-parameter R sufficiently large for the proximal point to exist; see the maximum of quadratic functions family in Section 4.

2.2 Algorithms Composing the Benchmark

As far as we can tell, CPROX is currently the only method specifically designed for proximal point calculations on nonconvex functions. Nevertheless, any general purpose NSO algorithm could be used to compute a proximal point for a nonconvex function. More precisely, since (1) is a nonconvex nonsmooth unconstrained optimization problem, the computation of the proximal point could be done by minimizing the function

$$f_R(\cdot) := f(\cdot) + R\frac{1}{2}|\cdot - x^0|^2.$$

Note that, having an oracle for f (i.e., computing $f(x)$ and $g(x) \in \partial f(x)$), gives in a straightforward manner an oracle for f_R , since for any $x \in \mathbb{R}^N$

$$f_R(x) = f(x) + R\frac{1}{2}|x - x^0|^2 \quad \text{and} \quad g_R(x) = g(x) + R(x - x^0) \in \partial f_R(x).$$

For our benchmark, in addition to CPROX, we numerically examine six NSO methods: N1CV2, N2FC1, PBUN, PVAR, PNEW, and RGS. We now give a brief description of the main features of the solvers tested.

2.3 Bundle Solvers N1CV2, N2FC1, PBUN, PVAR, PNEW

Bundle methods appear as a two step stabilization of cutting-planes methods. First, they generate a sampling point y^{j+1} by solving a stabilized quadratic programming subproblem. Second, they *select* some iterates satisfying certain descent condition. These selected points, or *stability centers*, form a subsequence $\{x^k\} \subset \{y^j\}$ that aims at guaranteeing a sufficient decrease in f_R . Specifically, given the last generated stability center x^k and $\mu_k > 0$, y^{j+1} is the solution to

$$\min_{y \in \mathbb{R}^N} \check{f}_R(y) + \frac{1}{2}\mu_k|y - x^k|^2,$$

where \check{f}_R is a cutting-planes model for the function f_R .

If for some $m_1 \in]0, 1[$

$$f_R(y^{j+1}) \leq f_R(x^k) - m_1 \left(f_R(x^k) - \check{f}_R(y^{j+1}) - \frac{1}{2}\mu_k|y^{j+1} - x^k|^2 \right),$$

then y^{j+1} provides a sufficient decrease. A new stability center is thus defined, $x^{k+1} = y^{j+1}$, μ_k is updated to μ_{k+1} , and both j and k are increased.

Otherwise, the model \check{f}_R is considered not accurate enough. Then a *null step* is declared: there is no new stability center and only j is increased.

Note that in both cases the new oracle information $f_R(y^{j+1})$ and $g_R^{j+1} \in \partial f_R(y^{j+1})$ is incorporated to the cutting-planes model \check{f}_R .

The first bundle algorithm we tested is N1CV2, the proximal bundle method from [11], designed along the lines described above for **convex** NSO. Parameters μ_k are updated with a “poor-man” rule that mimics a quasi-Newton method for minimizing the Moreau envelope of \tilde{f}_R . Since the method is designed for convex functions only, poor performance is to be expected for highly nonconvex functions. For this reason, N1CV2 is only used to benchmark the maximum of quadratic functions family, for which all the test functions are semi-convex; see Section 4.

The second bundle method is N2FC1, an ε -steepest descent bundle method specially designed for nonconvex minimization with box constraints (we set $|x_i| \leq 10^5$ in our runs). The method is described in [12]. It is essentially a dual method, following the lines given above for bundle methods. N2FC1 performs a line search that is shown to end after a finite number of steps for semismooth functions. Likewise, convergence of the method is shown for semismooth functions.

Bundle codes PBUN, PVAR, and PNEW are designed to find a close approximation to a local minimum of a nonlinear nonsmooth Lipschitzian function. PBUN is based on the proximal bundle method described in [13] using only first-order information. PVAR is a bundle method that uses variable metric techniques to estimate second-order information for the function; see [15]. Finally, PNEW is based on the bundle-Newton method [14], which uses (approximate) second-order information as a substitute of the Hessian matrix.

A Fortran 77 code for N1CV2 is available upon request at <http://www-rocq.inria.fr/estime/modulopt/optimization-routines/n1cv2.html>. N2FC1 can be requested from its author, Claude Lemaréchal, <http://www.inrialpes.fr/bipop/people/lemarechal/>. Fortran 77 source codes of PBUN, PVAR, and PNEW can be found at www.cs.cas.cz/~luksan/subroutines.html.

2.4 Algorithm CPROX

This algorithm, specifically designed to compute proximal points of nonconvex functions, was introduced in [8]. It is shown to be convergent for lower- \mathcal{C}^2 functions, whenever the prox-parameter R is sufficiently large.

Each iteration k of CPROX proceeds by splitting the proximal point parameter R into two values η_k and μ_k such that $\eta_k + \mu_k = R$. Rewriting the proximal point problem (1) as

$$\operatorname{argmin} \left\{ f(y) + R \frac{1}{2} |y - x^0|^2 \right\} = \operatorname{argmin} \left\{ \left(f(y) + \eta_k \frac{1}{2} |y - x^0|^2 \right) + \mu_k \frac{1}{2} |y - x^0|^2 \right\},$$

the idea of CPROX is, instead of working with the nonconvex function f , to work on the (hopefully more convex) function $f_{\eta_k} := f + \eta_k \frac{1}{2} |\cdot - x^0|^2$. Somewhat similarly to bundle methods, CPROX creates a piecewise linear model of f_{η_k} , and calculates the proximal point for this model by solving a quadratic programming problem. A subroutine for dynamically selecting η_k and μ_k ensures that these parameters will eventually satisfy the properties required for convergence of the algorithm. An additional feature of CPROX is that it checks that R is sufficiently large for (3) to hold as an equivalence.

A Matlab code for CPROX is available upon request from the authors.

2.5 Random Gradient Sampling Algorithm

The final algorithm we consider in this paper is the random gradient sampling algorithm RGS from [3]. Unlike the other methods examined in this work, RGS is not a bundle-like algorithm, but rather a stochastic method using ideas going back to [5] and [6]. Nevertheless, RGS is designed for dealing with oracle based minimization. Specifically, this method is intended for minimizing functions that are continuous and for which the gradient exists and is readily computable almost everywhere on \mathbb{R}^N . Algorithm RGS is largely inspired by the classical Clarke subgradient formula

$$\partial f_R(\bar{x}) = \limsup_{\varepsilon \searrow 0} \text{conv}\{\nabla f_R(x) : x \text{ such that } |x - \bar{x}| \leq \varepsilon\}.$$

Briefly, at iteration k , given x^k , the gradient of f_R is computed at x^k and at randomly generated points near x^k within a sampling diameter, and the convex combination of these gradients with smallest 2-norm, say d^k , is computed by solving a quadratic program. One should view $-d^k$ as a kind of stabilized steepest descent direction. A line search is then used to obtain the next iterate and reduce the function value. If the magnitude of d^k is below a prescribed tolerance, or a prescribed iteration limit is exceeded, the sampling diameter is reduced by a prescribed factor, and the process is repeated. Besides its simplicity and wide applicability, a particularly appealing feature of the gradient sampling algorithm is that it provides approximate “optimality certificates”: if d^k is small for a small sampling diameter, one can be reasonably sure that a local minimizer has been approximated.

Another interesting property of RGS is that, like bundle methods, it tends to detect nearby “ridges” and move parallel to them instead of towards them. Consider for example, the (convex) function $f(x_1, x_2) := \frac{1}{2}x_1^2 + 100|x_2|$ and the point $x^k := (1, 0.0001)$. The steepest descent at this point yields the unpromising search direction $-\nabla f(1, 0.0001) = (-1, -100)$. However, if the two points $(1, 0.0001)$ and $(1, -0.0001)$ are used to create a more robust local subgradient, namely,

$$\tilde{\partial}f(1, 0.0001) := \text{conv}\{\nabla f(1, 0.0001), \nabla f(1, -0.0001)\} = \text{conv}\{(1, 100), (1, -100)\},$$

then the new descent direction is $d^k = -\text{Proj}(0, \tilde{\partial}f(1, 0.0001)) = (-1, 0)$, a vastly more promising search direction. In this sense, the “robust local subgradient” $\tilde{\partial}f(x^k)$ can be interpreted as the nonconvex analog of the ε -subdifferential $\partial_\varepsilon f(x^k)$, a fundamental concept for convex bundle methods. It should be mentioned, however, that, contrary to modern bundle methods, RGS is a first order method which does not exploit any curvature information. This feature is confirmed by the fact that, for smooth functions, RGS is nothing but a gradient method.

A Matlab implementation of RGS is available at: www.cs.nyu.edu/faculty/overton/papers/gradsamp/.

3 Benchmark Parameters and Measures

All of the algorithms tested in this work have a variety of optional parameters which can be used to improve their performance. Accordingly,

in all of the tests performed, each algorithm’s input parameters were adjusted towards best performance.[‡]

[‡]The authors would like to again acknowledge Michael Overton’s assistance in setting the RGS parameters.

Completely fair data comparisons can only occur if all the solvers have a uniform stopping criterion. Since altering the actual solver source code is not practical (nor fair), it becomes difficult to meet such criterion. However, the codes for CPROX, N1CV2, and N2FC1 all have the option of setting a maximum number of oracle calls. Since we are considering proximal point computations in the setting of oracle based optimization, considering efficiency in terms of the number of function evaluations is natural. Therefore, we set the maximum number of oracle calls each algorithm is allowed to make and compare the “best found point” for each algorithm.

In all tests performed, each algorithm was set to terminate after 100 oracle calls were detected.

As mentioned, in the case of CPROX, N1CV2, and N2FC1 setting the maximum number of oracle calls permitted is a simple matter. Since CPROX makes exactly one oracle call per iteration, fixing the number of oracles is done by setting the maximum number of iterations to 100, while N1CV2 and N2FC1 have the option of setting a maximum number of oracle calls directly. In RGS, setting the maximum number of oracle calls permitted is not so simple. This is because RGS is equipped with various subroutines (such as random sampling and line searches) whose premature cessation may cause inaccurate test conclusions. Therefore, we allowed this algorithm to complete these subroutines before exiting and returning the final solution. In some cases this caused RGS to use more than 100 oracle calls. For the PBUN, PVAR, and PNEW codes, we used the built-in maximum function evaluation parameter (MFC). Like RGS, these codes do not stop immediately after 100 oracle calls have been completed, but instead stop after various subroutines complete. Sometimes, especially in higher dimensions, this can involve quite a few extra function calls.

A performance profile is essentially a condensed graphical form of statistical information, including solver robustness and efficiency. Our performance profiles use two ratios for measuring performance, depending on whether or not x^* , a *unique* solution to (1), is available.

If x^* is the known unique location of the proximal point and x^{best} is the best point found out of all the oracle calls made (i.e. x^{best} is such that, $f(x^{best}) + R\frac{1}{2}|x^{best} - x^0|^2 = \min_{i=0,1,2,\dots,n} \{f(x^i) + R\frac{1}{2}|x^i - x^0|^2\}$), our first formula

$$-\log_{10} \left(\frac{|x^{best} - x^*|}{|x^0 - x^*|} \right) \quad (\text{R.A.})$$

measures the relative gain in accuracy in computing the actual proximal point. We report these results via the \log_{10} scale, since the relative gain in accuracy can often be very large or very small. On this scale, a positive number (roughly) represents the number digits of accuracy obtained through the algorithm, a 0 means $x^{best} = x^0$, while a negative number means the algorithm actually moved away from the correct proximal point. It should be noted that movement away from the correct proximal point still requires $f(x^{best}) + R\frac{1}{2}|x^{best} - x^0| \leq f(x^0)$, so some improvement in the objective function must still have been achieved.

Performance profiles using the (R.A.) scale (Figure 1 in our benchmark) plot the (R.A.) value (on the x-axis) versus the portion of tests which successfully achieved this value (on the y-axis). As such, the location where a profile first decreases from the y value 1 describes the gain in accuracy the algorithm achieved on every problem, while the location where a profile first obtains a y value of 0 yields the best gain in accuracy achieved using that

algorithm. In general, algorithms whose profiles are “higher” have outperformed algorithms with “lower” profiles.

Formula (R.A.) provides information on quality of the solution returned by a solver, this is of particular interest for nonconvex problems. However, if x^* is unknown or not unique, this formula becomes unusable. In this case, we use a second formula, measuring the total decrease in the objective function:

$$f(x^0) - \left(f(x^{best}) + R \frac{1}{2} |x^{best} - x^0|^2 \right). \quad (\text{T.D.})$$

We no longer use a \log_{10} scale, nor do we divide by some factor to create a “relative total decrease”. The \log_{10} scaling is avoided because in the second formula the total decrease in objective function value is often quite small, so a \log_{10} scale would result in many negative numbers. This causes some conflict with our first formula where negative numbers are construed as bad. The reason we do not attempt to create a “relative total decrease” with the second formula is because the correct value to divide by would be $(f(x^0) - f(x^*) - R \frac{1}{2} |x^* - x^0|^2)$, but the (T.D.) scale is used in Section 5 where the actual proximal point is unknown and in Subsection 6.2 where the actual proximal point is very unlikely to be achieved.

Performance profiles using the (T.D.) scale plot, the (T.D.) value (on the x-axis) versus the portion of tests which successfully achieved this value (on the y-axis). As such, the location where a profile first decreases from the y value 1 describes the minimal decrease the algorithm achieved on every problem, while the location where a profile first obtains a y value of 0 yields the maximum decrease achieved using that algorithm. As before, in general, algorithms whose profiles are “higher” have outperformed algorithms with “lower” profiles.

4 “Maximum of Quadratics” Functions

In this section we consider test functions defined as the pointwise maximum of a finite collection of quadratic functions:

$$f(x) := \max \{ \langle x, A_i x \rangle + \langle B_i, x \rangle + C_i : i = 1, 2, \dots, \mathbf{nf} \}, \quad (4)$$

where A_i are $N \times N$ symmetric matrices (no assumption on their positive definiteness is made), $B_i \in \mathbb{R}^N$, and $C_i \in \mathbb{R}$.

This class of functions has several practical advantages. First, many different examples are easily created by choosing values for N and \mathbf{nf} , then randomly generating \mathbf{nf} objects A_i , B_i and C_i . Second, oracles are easy to define, because for each given x any *active* index $j \leq \mathbf{nf}$ (i.e., an index where the maximum in (4) is attained), yields a subgradient $A_j x + B_j \in \partial f(x)$. In particular, if there are \mathbf{nf}_{act} active indices at 0, the following relations hold (reordering indices if necessary):

$$f(0) = C_i = C \text{ for } i = 1, 2, \dots, \mathbf{nf}_{\text{act}} \quad \text{and} \quad \partial f(0) = \text{conv}\{B_i : i = 1, \dots, \mathbf{nf}_{\text{act}}\}. \quad (5)$$

Furthermore, a function given by (4) is always prox-bounded with computable threshold:

$$r_{pb} = \max\{|A_i| : i = 1, 2, \dots, \mathbf{nf}\}.$$

Thus, a “large enough” prox-parameter R can be estimated a priori, by taking any value bigger than the norm of all matrices A_i . However, it should be noted that with the application of this prox-parameter, the objective function for proximal point computation,

$f + R\frac{1}{2}|\cdot - x^0|^2$, is actually a convex function. Therefore, one would expect tried and true algorithms for convex optimization to vastly outperform any nonconvex optimization algorithm. The specialized solver N1CV2 was included in the benchmark for this particular family of test functions.

To assess our method, in all our runs we fix the desired proximal point to be the zero vector, and choose starting points x^0 so that $P_R f(x^0) = 0$. This is done as follows:

0. Select a lower and upper bound for random number generation $L, U \in \mathbb{R}$, and a prox-parameter scaling $R_{sc} \geq 1$.
1. Fix a dimension N , the number of quadratic functions defining the maximum \mathbf{nf} , and the number of active subgradients $\mathbf{nf}_{\text{act}} \leq \mathbf{nf}$ at $P_R(x^0) = 0 \in \mathbb{R}^N$.
2. Set $C_1 = C_2 = \dots = C_{\mathbf{nf}_{\text{act}}} = U$ and randomly generate $C_i \in]L, U[$ for $i = \mathbf{nf}_{\text{act}} + 1, \dots, \mathbf{nf}$.
3. For all $i \leq \mathbf{nf}$ randomly generate $B_i \in]L, U[^N$.
4. For all $i \leq \mathbf{nf}$ randomly generate symmetric matrices $A_i \in]L, U[^{N \times N}$ such that the minimum eigenvalue for each A_i is negative.
5. Set $R = R_{sc}(\max |A_i|) + 1$, take any $x^0 \in \frac{1}{R} \text{conv}\{B_i : i = 1, \dots, \mathbf{nf}_{\text{act}}\}$ as starting point.

The condition that the minimum eigenvalue for each A_i is negative in Step 4 ensures that the resulting function is nonconvex near 0. This step is easily accomplished by simply regenerating any A_i whose minimum eigenvalue is nonnegative (in practice, if $N > 1$, randomly generated symmetric matrices tend to result in at least one negative eigenvalue). In Step 5 we assure that $P_R f(x^0) = 0$ as,

$$p = P_R(x^0) \iff R(x^0 - p) \in \partial f(p) \quad \text{so,} \quad p = 0 \iff R x^0 \in \partial f(0).$$

For our purposes, in Step 0 we fix $L = -10$, $U = 10$ and $R_{sc} = 12$, then consider various combinations for N , \mathbf{nf} , and \mathbf{nf}_{act} .

For our benchmark, we use 8 different combinations of N , \mathbf{nf} , and \mathbf{nf}_{act} , then randomly generate 20 test functions for each combination. This provides a total of 160 test functions. We then ran each algorithm for a maximum of 100 oracle calls on each test function. Using formula (R.A.) we calculated the relative accuracy resulting for each test run. In Appendix A, Tables 1 and 2 report the worst accuracy, mean accuracy, and best accuracy for each test set and algorithm. Tables 1 and 2 also provide details on the combinations of N , \mathbf{nf} , and \mathbf{nf}_{act} used, and the mean number of oracle calls required for each test set and algorithm. Figure 1 provides a performance profile for the tests.

In general it would appear that CPROX outperformed the other algorithms in these tests. Although PNEW outperformed CPROX in many problems, it performed quite poorly on many others. It is worth noting that in test sets with $\mathbf{nf}_{\text{act}} = 1$, N2FC1 generally used only a fifth of the possible oracle calls, and in the best case provided excellent convergence. A similar effect can be seen in the PBUN, PVAR, and PNEW codes, where excellent convergence occurred in the cases of $\mathbf{nf}_{\text{act}} = 1$, but poor convergence occurred for $\mathbf{nf}_{\text{act}} > 1$. Algorithms CPROX, and N1CV2 also performed better on test sets with one active gradient, but not to the same degree.

Conversely, RGS behaved quite poorly on tests with only one active subgradient, but better on tests with multiply active subgradients. Since, when applied to a smooth function, RGS is nothing but a pure gradient method, poor performance in such cases is not surprising. In Sections 5 and 6 where the objective functions are less smooth we shall see RGS performance improve.

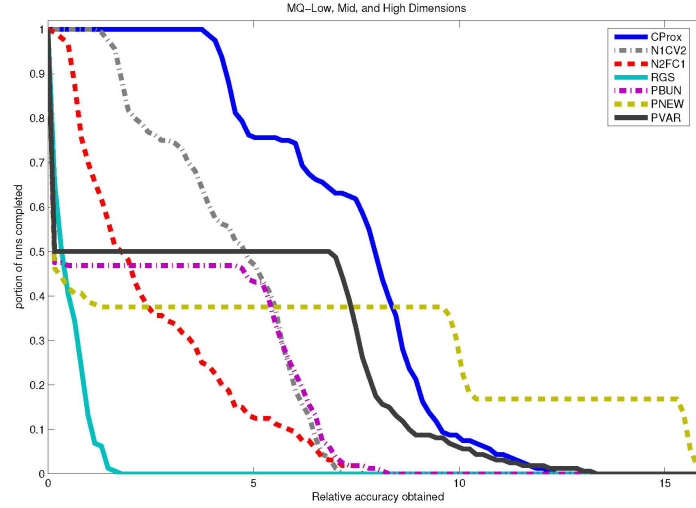


Figure 1: Performance Profile for maximum of quadratic functions family

5 “Piecewise Polynomials” Family

In this section we consider test sets constructed by composition of polynomial with absolute value functions. More precisely, we define

$$f(x) = \sum_{i=1}^N p_i(|x_i|),$$

where N is the dimension of the problem, and for $i = 1, \dots, N$, x_i is the i^{th} -coordinate of x , and p_i is a polynomial function.

As in Section 4, these functions are easy to generate, and have a simple subgradient formula

$$\partial f(x) = \partial p_1(|x_1|) \times \partial p_2(|x_2|) \times \dots \partial p_N(|x_N|).$$

Moreover, if we only use monic polynomials (polynomials where the coefficient of the highest power is equal to 1), we ensure that the function is bounded below and therefore prox-bounded. Finally, away from the origin, such functions can easily be rewritten as the maximum of polynomials, and therefore, are lower- \mathcal{C}^2 (see [20, Def 10.29 & Thm 10.33]).

However, considering the one dimensional example

$$f(x) = (|x| - 1)^2 - 1 = |x|^2 - 2|x|,$$

it is clear that functions of this form need not be regular at 0. Furthermore, whenever the degree of the polynomials used is greater than two, such functions are not semi-convex.

A disadvantage of such functions is the difficulty in computing the correct proximal point. Although for points away from the origin, with sufficiently large R , a closed form solution for the proximal point could be constructed, the solution would be nontrivial. Moreover, due to the symmetry of the function, the proximal point at 0 is not unique. Indeed, suppose

$p \in P_R f(0)$, so $f(p) + R_{\frac{1}{2}}|p - 0|^2 = \min\{f(y) + R_{\frac{1}{2}}|y - 0|^2\}$. Since $f(p) = f(-p)$ and $R_{\frac{1}{2}}|p - 0|^2 = R_{\frac{1}{2}}|-p - 0|^2$ we must also have $-p \in P_R f(0)$. For these reasons, we chose to use the second of our comparison formulae, (T.D.).

We generate our test functions as follows:

0. Select a lower and upper bound for random number generation $L, U \in \mathbb{R}$.
1. Fix a dimension N , and the degree for the polynomials **Deg**.
2. For each $i = 1, 2, \dots, N$ generate **Deg** - 1 random numbers $a_1, a_2, \dots, a_{\text{Deg}-1} \in [L, U]$ and set

$$p_i(x) = x^{\text{Deg}} + \sum_{j=1}^{\text{Deg}-1} a_j x^j.$$

3. Set $f(x) = \sum_{i=1}^N p_i(|x_i|)$.

For our purposes, in Step 0 we fix L and U to be -1 and 0 . This causes a nonconvex movement in all polynomial powers of degree less than **Deg**. Notice that we have also set the constant coefficient for each polynomial, a_0 , to be zero. Since the polynomials are summed up, a positive constant term would have no effect, other than a vertical shifting of the function.

Having generated our test problems, we consider various prox-parameters and two different prox-centers for the calculations.

For our first prox-center, we select a location where the function should behave smoothly: $x_i^0 = 1$ for all $i = 1, \dots, N$. Our second prox-center focuses on an area where the function is very poorly behaved: $x_i^0 = 0$ for all $i = 1, \dots, N$. Since the generated test functions are not prox-regular (nor even regular) at $0 \in \mathbb{R}^N$, we anticipate a problem in the CPROX code in this case. Specifically, CPROX involves a subroutine which checks whether R is sufficiently large to ensure that the statement (3) holds as an if-and-only-if statement. Since the function is not prox-regular at our prox-center, we must expect this subroutine to fail. To examine the effect this subroutine has on convergence, we consider a second version of the algorithm denoted CPRFR, which is identical to CPROX, except that it bypasses this subroutine (this is done by setting the parameter “P.CheckR” to “off” in the CPROX code).

The convex solver N1CV2 was excluded from the benchmark for this (highly nonconvex) family of functions. In Appendix A, in Tables 3 to 5 we compare the total decrease in objective function value for each algorithm, using the prox-center $x_i^0 = 1$. Tables 3 to 5 also provide details on the combinations of N , **Deg**, and R used, and the mean number of oracle calls required for each test set and algorithm. Likewise, Tables 6 to 8 provide the same data with the prox-center $x_i^0 = 0$ for $i = 1, \dots, N$. To ease the reading in all tables, (T.D.) values of less than 10^{-6} are reported as 0.

Figures 2 and 3 provide a performance profiles for the two different starting points.

Surprisingly, regardless of the starting point, CPROX and CPRFR caused nearly identical total decrease in objective function value. However, CPROX did tend to terminate earlier. This suggests that the subroutine which determined if R is sufficiently large did not fail until CPROX had already come quite close to a local minimum.

In general it would appear that when starting at $x_i^0 = 1$ CPROX (and CPRFR) outperformed the remaining algorithms, with RGS coming a reasonably close second. Although all algorithms performed well when **Deg** was set to 3, when **Deg** ≥ 5 only CPROX (CPRFR) and RGS succeeded in generating an improvement in objective function value. N2FC1 fails

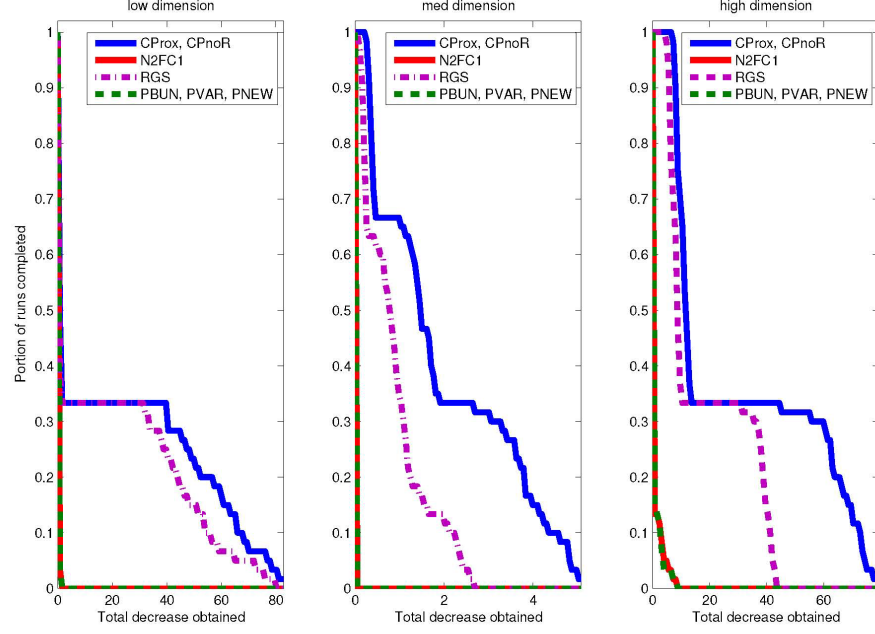


Figure 2: Performance Profiles for piecewise polynomials family ($x_i^0 = 1$)

mostly in the quadratic program solution, perhaps tighter box constraints would improve N2FC1 performance in these tests.

Shifting the prox-center to the nonregular point $0 \in \mathbb{R}^N$ seemed to have little effect on CPROX, CPRFR, and RGS. Again we see these algorithms performing well, with RGS only slightly outperformed by CPROX. However, the nonregular prox-center seemed to have a positive effect for all the remaining algorithms. This can probably be explained by the fact that, when beginning at a strictly nonconvex point any search direction will likely provide some decrease in function value.

6 Two Specific Test Functions

In this section we consider the following functions:

1. The Spike:

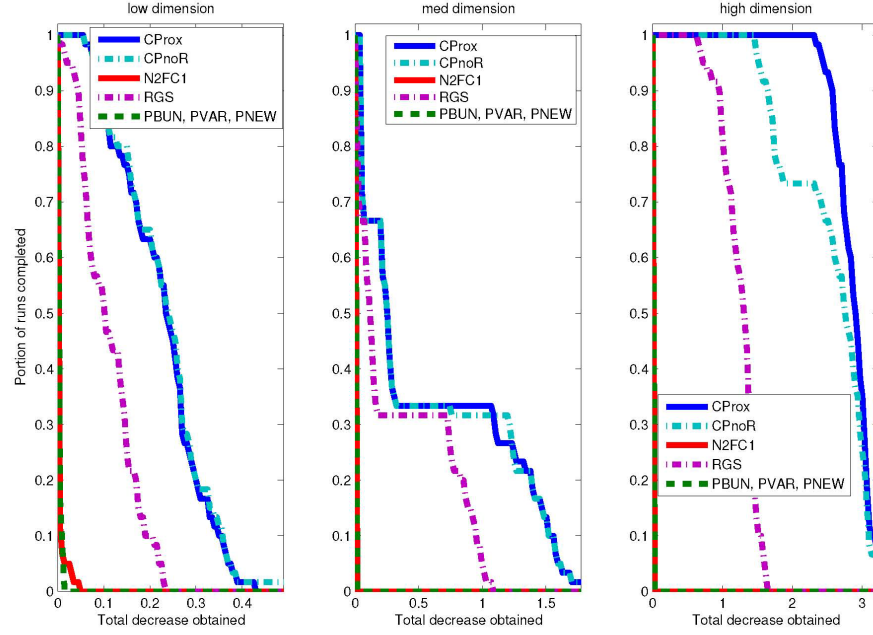
$$f := \sqrt{|x|},$$

2. The Waterfalls: for $x_i \in \mathbb{R}$ set

$$f_i(x_i) := \begin{cases} x_i^2 & \text{if } x_i \leq 0 \\ -2^{-n+1}(x_i - 2^n)^2 + 2^n & \text{if } x_i \in [2^{n-1}, 2^n] \text{ for some } n, \end{cases}$$

then, for $x = (x_1, \dots, x_N) \in \mathbb{R}^N$, set

$$f(x) = \sum_i f_i(x_i).$$

Figure 3: Performance Profiles for piecewise polynomials family ($x_i^0 = 0$)

6.1 The Spike

The function $f = \sqrt{|\cdot|}$ is the classical example of a regular function which is not prox-regular at 0. As such, f is not semi-convex, nor lower- \mathcal{C}^2 near 0. Therefore, one cannot rely on the classical subgradient formula from equation (3). Conversely, the function is bounded below, and therefore prox-bounded with threshold equal to 0. Moreover, as we shall see below, numerically calculating an exact proximal point is achievable for this problem.

Due to the highly nonconvex nature of this problem, N1CV2 was excluded from the benchmark for this problem.

Before we discuss the prox-parameters and prox-centers we test, it is worth noting that, due to the radial symmetry this function exhibits, for CPROX, N2FC1, PBUN, PVAR, and PNEW we need to consider this function only in one dimension. Indeed, consider a prox-center $x^0 \neq 0$, prox-parameter $R > 0$ and any point x^m on the line connecting $0 \in \mathbb{R}^N$ to x^0 ; i.e., $x^m = \tau x^0$ for some $\tau \in \mathbb{R} \setminus \{0\}$. The subdifferential of $f + R\frac{1}{2}|\cdot - x^0|^2$ is given by

$$\partial(\sqrt{|\cdot|} + R\frac{1}{2}|\cdot - x^0|^2)(x^m) = \frac{1}{2}|x^m|^{-3/2}x^m + R(x^m - x^0).$$

Since $x^m \neq 0$, we see that

$$\nabla f_R(x^m) = \left(\frac{1}{2}\tau^{-3/2}|x^0|^{-3/2} + R(\tau - 1) \right) x^0.$$

In other words, all the cutting planes created when calling the oracle will act only along the

direction created by x^0 . As such, all iterates created via CPROX, N2FC1, PBUN, PVAR, and PNEW will lie on the line connecting x^0 and 0.

This analysis fails to hold for RGS, as the random sampling involved can move iterates away from the line segment. We therefore test CPROX, N2FC1, PBUN, PVAR, and PNEW in one dimension only, but test RGS in dimensions 1, 10, and 100. Furthermore, to examine the randomness invoked by RGS, for each test run of RGS we run the algorithm 20 times. Despite its random nature, RGS was remarkably consistent in its results (the standard deviations for the 20 runs for each test was always less than 10^{-4}). In the results below we always select the best resulting run for RGS.

For our tests we set the prox-center to be $x^0 = 1$ (in higher dimensions we take $x_1^0 = 1$, and $x_i^0 = 0$ for $i = 2, \dots, N$), and consider various prox-parameters: $R = 0.1$, $R = 0.25$, $R = 1$, and $R = 2$. Even though the classical subgradient formula from equation (3) does not hold with equality for this function, we can nonetheless use it to derive the critical points for the function:

$$\begin{aligned} p = P_R f(x^0) \Rightarrow 0 \in \partial \left(\sqrt{|\cdot|} + R \frac{1}{2} |\cdot - x^0|^2 \right) (p), \\ p = P_R f(x^0) \Rightarrow p = 0 \text{ or } \frac{1}{2} |p|^{-3/2} p + R(p - x^0) = 0. \end{aligned} \quad (6)$$

This equation can be solved symbolically by any number of mathematical solvers. Using Maple we found:

- for $R = 0.1$, $R = 0.25$, and $R = 1$ equation (6) yields two imaginary roots and the correct proximal point $p = 0$,
- for $R = 2$ equation (6) yields $p \in \{0, 0.0726811601, 0.7015158585\}$, the correct proximal point is $p = 0.7015158585$.

As for Section 4, we report the accuracy as obtained via equation (R.A.). Similarly to Section 5, since f is not semi-convex, we also include results of the algorithm CPRFR. For the same reason, the convex solver N1CV2 was excluded from the benchmark. Our results are listed in Appendix A, Table 9.

Our first result of note is the appearance of “ ∞ ” in the one dimensional RGS tests. This means that RGS exactly identified the correct proximal point of the function. Although this looks impressive, it is likely due to RGS’s line-search style, and the fact the prox-center is exactly 1 unit away from the proximal point. If other design factors were involved, the “ ∞ ” accuracy would likely reappear in the higher dimension tests.

Our next observation is that the results of CPROX, CPRFR, and N2FC1 appear to rely heavily on the prox-parameter. PBUN, PVAR, and PNEW also fluctuated with respect to the prox-parameter, but to a lesser degree. Conversely, RGS resulted in a consistent performance regardless of the prox-parameter. This suggests that, although in general RGS may not be the most successful algorithm, it is the most robust. Since robustness was a major concern in the design of RGS, this is a very positive result for its authors.

In comparing CPROX and CPRFR we note that the removal of the subroutine for determining if R is sufficiently large to guarantee convergence does not appear to have any positive effect on the algorithm. Indeed in the one case where CPRFR outperformed CPROX ($R = 2$), CPRFR used 100 times more oracles calls than CPROX, but only improved (R.A.) by a factor of 7.22.

6.2 The Waterfalls

The waterfalls function is also poorly behaved in terms of proximal envelopes. Although the function is prox-bounded with threshold equal to 0, it is not semi-convex, nor lower- \mathcal{C}^2

near 0, nor prox-regular at $0 \in \mathbb{R}^N$. Therefore, once more one cannot rely on the classical subgradient formula from equation (3).

For example, consider a point $\bar{x} \in \mathbb{R}^N$ such that $\bar{x}_i = 2^{n_i}$ for each $i = 1, 2, \dots, N$. The subgradient of waterfalls function is easily seen to be

$$\partial f(x) = \partial f_1(x_1) \times \partial f_2(x_2) \times \dots \times \partial f_N(x_N),$$

where

$$\partial f_i(x_i) = \begin{cases} [0, 2] & : x_i = 2^n \text{ for some } n \\ -2^{n+2}(x_i - 2^n) & : x_i \in [2^{n-1}, 2^n]. \end{cases}$$

So, at \bar{x} , we see $\partial f(\bar{x}) = [0, 2] \times [0, 2] \times \dots [0, 2]$. Hence, if \bar{x} is the prox-center, then for any $R > 0$ one has

$$p = \bar{x} \text{ satisfies } 0 \in \partial f(p) + R(p - \bar{x}).$$

However, unless R is large, $\bar{x} \neq P_R f(\bar{x})$.

Moreover, consider the point $\bar{x}/2^m$. Then

$$\partial f(\bar{x}/2^m) + R(\bar{x}/2^m - \bar{x}) = ([0, 2] \times [0, 2] \times \dots [0, 2]) - (1 - 2^{-m})R\bar{x}.$$

If $R\bar{x}_i < 2$ for each $i = 1, 2, \dots, N$, then this will result in $0 \in \partial f(\bar{x}/2^m) + R(\bar{x}/2^m - \bar{x})$ for all m sufficiently large. That is, equation (3) will be satisfied at an infinite number of points. In this case the correct proximal point is 0.

For our tests, we initially considered the prox-center $x_i^0 = 1$ for $i = 1, \dots, N$, several values of R and several dimensions. However, in this case all algorithms tested stalled immediately. This is probably because all the algorithms tested contain some stopping criterion based on (a convex combination of) subgradients approaching zero.

We therefore considered what would happen if instead we randomly selected a prox-center nearby $x_i^0 = 1$, but not exactly equal to this point. For each test, we randomly generated 20 prox-centers of the form

$$x_i^0 = \mathbf{rand}_i \quad \text{for } i = 1, \dots, N,$$

where \mathbf{rand}_i is a random number with values in $[1 - 0.0001, 1 + 0.0001]$.

The convex solver N1CV2 was excluded from the benchmark for this (highly nonconvex) family of functions. We ran each algorithm starting from this prox-center, and compared the improvement in objective function decrease via formula (T.D.). Our results appear in Appendix A, Tables 10 to 12, and in the performance profile in Figure 4.

As in Subsection 6.1, drawing conclusions from these test results is tricky. In low and medium dimensions, all algorithms performed fairly well, with perhaps RGS yielding a slight advantage in (T.D.). In high dimensions, this advantage shifts to CPROX and CPRFR.

In ten dimensions, we see a fairly even (T.D.) in all algorithms, however, CPROX uses less oracle calls than the remaining solvers.

In 100 dimensions, we see CPROX and CPRFR giving considerably better (T.D.) than the remaining algorithms, while continuing to use a small number of oracle calls. Nonetheless, we see RGS continues to perform adequately, reinforcing the robustness property of this algorithm. Of concern is the number of oracle calls used by PNEW, suggesting that PNEW is not a robust algorithm for high dimension calculations (at least not with the best parameter setting we were able to find).

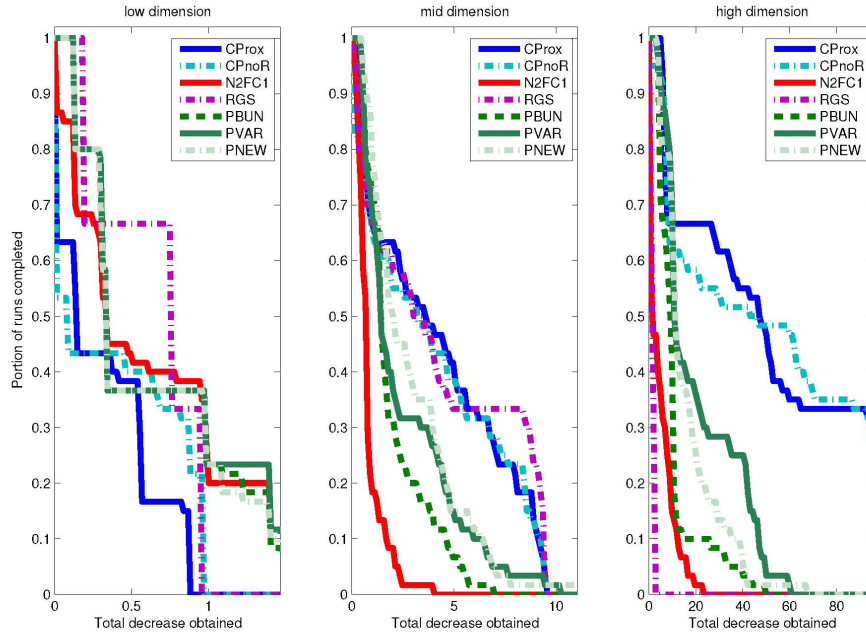


Figure 4: Performance Profiles for waterfalls family

7 Conclusions

In this paper we have presented several new benchmarking algorithmic techniques for computing proximal points of nonconvex functions. The choice of test problems for benchmarks is difficult and inherently subjective. Therefore, in order to reduce the risk of bias in benchmark results, it is sometimes helpful to perform benchmarks on various sets of models and observing solver performance trends over the whole rather than relying on a single benchmark test set. For this reason, in Sections 4 and 5 we give two methods to randomly generate large collections of challenging test functions for problem (1), while in Section 6 we outlined two very specific test functions which should be of interest to nonconvex optimization benchmarking.

Using these problems, this paper compares seven algorithms: CPROX, N1CV2, N2FC1, RGS, PBUN, PVAR, and PNEW. We further examined two versions of CPROX, to determine (numerically) if a small modification of the code would improve performance.

One can cautiously draw the following conclusions concerning the codes. Overall we saw CPROX in general outperformed the remaining algorithms *on this collection of test problems*. N1CV2 has proved its value for the “more convex” instances, i.e., the maximum of quadratic functions family. RGS generally performed quite well, especially in the high dimension and nonsmooth nonconvex cases, reinforcing the robustness of that particular algorithm. Finally, PBUN, PVAR, and PNEW appear most suited for problems with a low level of nonsmoothness.

References

- [1] S.C. Billups, S.P. Dirkse, and M.C. Ferris, A comparison of large scale mixed complementarity problem solvers, *Comput. Optim. Appl.* 7 (1997) 3–25.
- [2] J.F. Bonnans, J.C. Gilbert, C. Lemaréchal, and C. Sagastizábal, *Numerical Optimization. Theoretical and Practical Aspects*, Universitext. Springer-Verlag, Berlin, 2003.
- [3] J.V. Burke, A.S. Lewis, and M.L. Overton, Approximating subdifferentials by random sampling of gradients, *Math. Oper. Res.* 27(3) (2002) 567–584.
- [4] E.D. Dolan and J.J. Moré, Benchmarking optimization software with performance profiles, *Math. Program.* 91(2, Ser. A) (2002) 201–213.
- [5] A. A. Goldstein, Optimization of Lipschitz continuous functions, *Math. Program.* 13(1) (1977) 14–22.
- [6] A.M. Gupal, A method for the minimization of almost differentiable functions. (Russian. English summary), *Kibernetika (Kiev)* 1 (1977) 114–116. English translation: *Cybernetics* 13(1) (1977) 115–117.
- [7] W.L. Hare and R.A. Poliquin, Prox-regularity and stability of the proximal mapping, *J. Convex Anal.*, accepted for publication, (2006).
- [8] W.L. Hare and C. Sagastizábal, Computing proximal points of nonconvex functions, *Math. Program.* (Ser. B), accepted for publication, (2006).
- [9] J.-B. Hiriart-Urruty, Miscellanies on nonsmooth analysis and optimization, in *Nondifferentiable optimization: motivations and applications (Sopron, 1984)*, volume 255 of *Lecture Notes in Econom. and Math. Systems*, Springer, Berlin, 1985, pp. 8–24.
- [10] K.C. Kiwiel, Proximity control in bundle methods for convex nondifferentiable minimization, *Math. Program.* 46 (1990) 105–122.
- [11] C. Lemaréchal and C. Sagastizábal, Variable metric bundle methods: from conceptual to implementable forms, *Math. Program.* 76 (1997) 393–410.
- [12] C. Lemaréchal, J.-J. Strodiot, and A. Bihain, On a bundle method for nonsmooth optimization, in *Nonlinear Programming 4*, O.L. Mangasarian, R.R. Meyer and S.M. Robinson (eds.), Academic Press, 1981, pp. 245–282.
- [13] L. Lukšan, Bundle algorithms for nonsmooth unconstrained minimization, Research Report V-608, Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague, Czech Republic, 1995.
- [14] L. Lukšan and J. Vlček, A bundle-Newton method for nonsmooth unconstrained minimization, *Math. Program.* 83(3, Ser. A) (1998) 373–391.
- [15] L. Lukšan and J. Vlček, Globally convergent variable metric method for convex nonsmooth unconstrained minimization. *J. Optim. Theory Appl.* 102 (3) (1999) 593–613,
- [16] L. Lukšan and J. Vlček, NDA: Algorithms for nondifferentiable optimization, Research Report V-797, Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague, Czech Republic, 2000.

- [17] R. Mifflin, Semi-smooth and semi-convex functions in constrained optimization. *SIAM J. Control Optim.* 15 (1997) 959–972.
- [18] R. Mifflin, A modification and extension of Lemarechal’s algorithm for nonsmooth minimization, *Math. Program. Stud.* 17 (1982) 77–90.
- [19] J-J. Moreau, Proximité et dualité dans un espace Hilbertien, *Bull. Soc. Math. France* 93 (1965) 273–299.
- [20] R.T. Rockafellar and R.J-B. Wets, *Variational Analysis*, Springer-Verlag, Berlin, 1998.

Manuscript received 1 March 2006
revised 25 May 2006
accepted for publication 25 May 2006

WARREN HARE

Dept. of Comp. and Software, McMaster University, Hamilton, ON, Canada
E-mail address: `whare@cecm.sfu.ca`

CLAUDIA SAGASTIZÁBAL

IMPA, Estrada Dona Castorina 110, Jardim Botânico, Rio de Janeiro RJ 22460-320, Brazil
E-mail address: `sagastiz@impa.br`

A Appendix: Tables.

$N, \text{nf}, \text{nf}_{\text{act}}$	Alg.	Worst (R.A.)	Mean (R.A.)	Best (R.A.)	Mean Oracle Calls
5, 5, 1	CProX	7.4	8.2	9.5	100
	N1CV2	5.1	5.9	7.0	100
	N2FC1	0.1	3.6	7.6	20
	RGS	0.051	0.68	1.7	70
	PBUN	5.4	5.9	7.1	43
	PVAR	7.2	9.6	13	77
	PNEW	9.6	11	16	77
10, 5, 5	CProX	6.0	6.3	7.2	100
	N1CV2	3.2	3.6	4.2	100
	N2FC1	1.1	3.7	5.5	86
	RGS	0.029	0.31	0.63	114
	PBUN	-0.46	-0.19	0.13	44
	PVAR	-0.46	-0.19	0.13	61
	PNEW	-0.46	-0.19	0.13	113
20, 30, 1	CProX	7.5	8.4	9.7	100
	N1CV2	4.0	5.5	7.0	100
	N2FC1	0.25	2.6	7.0	18
	RGS	0.25	0.73	1.4	101
	PBUN	5.3	6.0	7.1	47
	PVAR	7.0	8.1	11	79
	PNEW	9.8	13	16	224
20, 30, 30	CProX	8.4	10	12	100
	N1CV2	3.1	4.7	6.5	100
	N2FC1	0.88	1.6	2.6	101
	RGS	-0.012	0.15	0.32	123
	PBUN	-0.81	-0.66	-0.49	46
	PVAR	-0.81	-0.66	-0.49	89
	PNEW	-0.81	-0.66	-0.49	227

Table 1: Low Dimension Test Results for Maximum of Quadratics

$N, \text{nf}, \text{nf}_{\text{act}}$	Alg.	Worst (R.A.)	Mean (R.A.)	Best (R.A.)	Mean Oracle Calls
50, 30, 1	CProX	7.8	8.5	9.3	96
	N1CV2	3.5	5.8	7.2	100
	N2FC1	1.1	3.7	7.6	23
	RGS	0.57	0.87	1.4	156
	PBUN	-6.9	3.9	8.3	67
	PVAR	6.9	7.8	10	98
	PNEW	9.7	13	15	481
50, 60, 30	CProX	3.9	4.6	5.0	100
	N1CV2	1.2	1.8	2.6	100
	N2FC1	0.48	0.73	0.90	101
	RGS	-0.054	0.056	0.12	121
	PBUN	-2.1	-0.96	-0.61	89
	PVAR	-0.76	-0.67	-0.60	98
	PNEW	-0.76	-0.67	-0.60	471
100, 30, 1	CProX	7.6	8.3	9.4	100
	N1CV2	4.4	5.9	6.9	100
	N2FC1	1.0	3.4	7.5	28
	RGS	0.65	0.91	1.1	107
	PBUN	5.5	6.4	7.3	59
	PVAR	7.0	7.5	8.7	99
	PNEW	-3.3	0.28	1.3	4517
100, 30, 30	CProX	3.9	4.3	5.6	100
	N1CV2	1.4	1.9	2.7	100
	N2FC1	0.47	0.64	0.84	101
	RGS	0.0032	0.022	0.044	111
	PBUN	-0.73	-0.67	-0.60	56
	PVAR	-0.73	-0.67	-0.60	102
	PNEW	-0.70	-0.52	-0.22	4677

Table 2: Mid and High Dimension Test Results for Maximum of Quadratics

N, Deg, R	Alg.	Minimal (T.D.)	Mean (T.D.)	Maximal (T.D.)	Mean Oracle Calls
5, 3, 10	CProX	0.285	0.517	0.943	92.4
	CPrFR	0.285	0.517	0.943	96.3
	N2FC1	0	0	1.157	60.1
	RGS	0.009	0.302	0.450	71.4
	PBUN	0	0	1.165	104.0
	PVAR	0	0	1.175	114.1
	PNEW	0	0	1.138	132.5
10, 5, 10	CProX	0.368	0.891	1.681	87.3
	CPrFR	0.368	0.891	1.681	100.0
	N2FC1	0	0	0	54.0
	RGS	0.108	0.551	1.150	89.6
	PBUN	0	0	0	103.0
	PVAR	0	0	0	117.3
	PNEW	0	0	0	175.4
20, 9, 20	CProX	39.837	59.538	83.672	100.0
	CPrFR	39.837	59.538	83.672	100.0
	N2FC1	0	0	0	93.3
	RGS	31.888	50.216	79.494	124.3
	PBUN	0	0	0	104.8
	PVAR	0	0	0	114.6
	PNEW	0	0	0	441.2

Table 3: Low Dimension Test Results for Piecewise Polynomials: $x_i^0 = 1$

N, Deg, R	Alg.	Minimal (T.D.)	Mean (T.D.)	Maximal (T.D.)	Mean Oracle Calls
50, 5, 10	CProX	2.697	4.034	5.138	91.7
	CPrFR	2.697	4.034	5.138	100.0
	N2FC1	0	0	0	45.0
	RGS	0.184	1.492	2.687	153.9
	PBUN	0	0	0	104.3
	PVAR	0	0	0	116.7
	PNEW	0	0	0	440.4
50, 5, 50	CProX	1.009	1.491	1.872	35.5
	CPrFR	0.882	1.362	1.872	100.0
	N2FC1	0	0	0	86.8
	RGS	0.428	0.913	1.254	156.0
	PBUN	0	0	0	104.7
	PVAR	0	0	0	115.8
	PNEW	0	0	0	398.4
50, 5, 250	CProX	0.243	0.357	0.445	67.1
	CPrFR	0.243	0.357	0.445	95.3
	N2FC1	0	0	0	58.6
	RGS	0.069	0.189	0.272	158.0
	PBUN	0	0	0	101.8
	PVAR	0	0	0	116.0
	PNEW	0	0	0	403.4

Table 4: Medium Dimension Test Results for Piecewise Polynomials: $x_i^0 = 1$

N, Deg, R	Alg.	Minimal (T.D.)	Mean (T.D.)	Maximal (T.D.)	Mean Oracle Calls
100, 3, 10	CProX	10.282	11.810	13.567	48.5
	CPrFR	6.952	11.381	13.567	100.0
	N2FC1	0	0.116	8.225	43.4
	RGS	7.674	8.755	9.942	103.0
	PBUN	0	0.199	8.225	107.2
	PVAR	0	0.199	8.225	119.0
	PNEW	0	0.199	8.226	700.2
100, 5, 10	CProX	6.644	8.477	10.493	91.7
	CPrFR	6.644	8.477	10.493	100.0
	N2FC1	0	0	0	44.0
	RGS	4.454	6.253	8.040	103.0
	PBUN	0	0	0	104.7
	PVAR	0	0	0	117.5
	PNEW	0	0	0	860.2
100, 7, 10	CProX	44.892	66.890	80.046	100.0
	CPrFR	44.892	66.890	80.046	100.0
	N2FC1	0	0	0	42.1
	RGS	30.836	39.375	43.354	103.0
	PBUN	0	0	0	105.1
	PVAR	0	0	0	112.9
	PNEW	0	0	0	1509.2

Table 5: High Dimension Test Results for Piecewise Polynomials: $x_i^0 = 1$

N, Deg, R	Alg.	Worst (T.D.)	Mean (T.D.)	Best (T.D.)	Mean Oracle Calls
5, 3, 10	CProX	0.058	0.140	0.288	90.1
	CPrFR	0.058	0.142	0.288	100.0
	N2FC1	0	0.007	0.044	63.4
	RGS	0.019	0.076	0.175	67.3
	PBUN	0	0.003	0.012	103.8
	PVAR	0	0.004	0.019	115.5
	PNEW	0	0.003	0.029	138.8
10, 5, 10	CProX	0.133	0.278	0.426	65.4
	CPrFR	0.150	0.289	0.495	100.0
	N2FC1	0	0.004	0.020	55.6
	RGS	0.051	0.159	0.234	76.0
	PBUN	0	0.004	0.015	103.8
	PVAR	0	0.004	0.019	114.1
	PNEW	0	0.004	0.015	169.1
20, 9, 20	CProX	0.156	0.257	0.390	100.0
	CPrFR	0.156	0.257	0.390	100.0
	N2FC1	0	0	0.006	40.0
	RGS	0.004	0.094	0.186	96.3
	PBUN	0	0	0.003	106.6
	PVAR	0	0	0.003	117.1
	PNEW	0	0	0.003	226.5

Table 6: Low Dimension Test Results for Piecewise Polynomials: $x_i^0 = 0$

N, Deg, R	Alg.	Worst (T.D.)	Mean (T.D.)	Best (T.D.)	Mean Oracle Calls
50, 10, 5	CProX	1.073	1.410	1.795	36.0
	CPrFR	0.759	1.406	1.795	100.0
	N2FC1	0	0	0.002	82.9
	RGS	0.096	0.843	1.075	154.8
	PBUN	0	0	0.002	105.0
	PVAR	0	0	0.002	118.9
	PNEW	0	0	0.002	418.3
50, 10, 50	CProX	0.205	0.249	0.313	100.0
	CPrFR	0.205	0.249	0.313	100.0
	N2FC1	0	0	0.001	18.8
	RGS	0.073	0.119	0.180	157.0
	PBUN	0	0	0	105.6
	PVAR	0	0	0	115.8
	PNEW	0	0	0	417.3
50, 10, 250	CProX	0.040	0.049	0.061	100.0
	CPrFR	0.040	0.049	0.061	100.0
	N2FC1	0	0	0	41.9
	RGS	0	0.013	0.028	159.3
	PBUN	0	0	0	102.0
	PVAR	0	0	0	113.7
	PNEW	0	0	0	404.4

Table 7: Mid Dimension Test Results for Piecewise Polynomials: $x_i^0 = 0$

N, Deg, R	Alg.	Worst (T.D.)	Mean (T.D.)	Best (T.D.)	Mean Oracle Calls
100, 3, 10	CProX	2.340	2.787	3.218	24.3
	CPrFR	1.564	2.539	3.218	100.0
	N2FC1	0	0	0	45.0
	RGS	0.630	1.111	1.554	103.0
	PBUN	0	0	0.001	105.3
	PVAR	0	0	0.001	123.5
	PNEW	0	0	0.001	758.5
100, 5, 10	CProX	2.544	2.905	3.246	29.4
	CPrFR	1.473	2.604	3.246	100.0
	N2FC1	0	0	0	45.0
	RGS	0.936	1.292	1.628	103.0
	PBUN	0	0	0.001	105.3
	PVAR	0	0	0.001	121.8
	PNEW	0	0	0.002	703.6
100, 7, 10	CProX	2.581	2.912	3.266	14.9
	CPrFR	1.474	2.487	3.121	100.0
	N2FC1	0	0	0	45.0
	RGS	0.970	1.332	1.642	103.0
	PBUN	0	0	0.001	105.4
	PVAR	0	0	0.001	121.4
	PNEW	0	0	0.001	808.8

Table 8: High Dimension Test Results for Piecewise Polynomials: $x_i^0 = 0$

R, N	Alg.	(R.A.)	Oracle Calls	R, N	Alg.	(R.A.)	Oracle Calls
0.1, 1	CProX	0.176	2	1, 1	CProX	0.301	1
	CPrFR	-0.106	100		CPrFR	0.15	100
	N2FC1	0	2		N2FC1	1.08	5
	RGS	∞	5		RGS	∞	5
	PBUN	3.26	100		PBUN	5.28	100
	PVAR	2.18	115		PVAR	2.02	133
	PNEW	6.47	148		PNEW	7.32	149
0.1, 10	RGS	4.89	124	1, 10	RGS	5.02	126
0.1, 100	RGS	3.80	103	1, 100	RGS	3.62	103
0.25, 1	CProX	15.477	2	2, 1	CProX	1.31	1
	CPrFR	15.477	3		CPrFR	9.47	100
	N2FC1	1.27	12		N2FC1	0.186	100
	RGS	∞	5		RGS	0.696	57
	PBUN	3.70	100		PBUN	3.70	109
	PVAR	1.44	104		PVAR	1.44	120
	PNEW	7.65	137		PNEW	0.17	108
0.25, 10	RGS	4.88	124	2, 10	RGS	4.81	124
0.25, 100	RGS	3.81	103	2, 100	RGS		104

Table 9: Test Results for the Spike

R, dim	Alg.	Minimal (T.D.)	Mean (T.D.)	Maximal (T.D.)	Mean Oracle Calls
0.1, 1	CProX	0	0.360	0.555	4.3
	CPrFR	0	0.631	0.970	100.0
	N2FC1	0	0.771	1.481	55.5
	RGS	0.950	0.950	0.951	63.5
	PBUN	0.332	0.708	1.479	100.4
	PVAR	0.330	0.720	1.479	101.5
	PNEW	0.334	0.736	1.479	118.6
0.5, 1	CProX	0	0.493	0.876	26.8
	CPrFR	0	0.481	0.877	100.0
	N2FC1	0	0.627	1.398	46.9
	RGS	0.750	0.750	0.751	65.3
	PBUN	0.299	0.668	1.396	101.0
	PVAR	0.285	0.634	1.396	101.3
	PNEW	0.299	0.684	1.396	120.0
2.5, 1	CProX	0	0.082	0.141	1.6
	CPrFR	0	0.030	0.089	100.0
	N2FC1	0.049	0.409	0.980	51.9
	RGS	0.188	0.188	0.188	66.6
	PBUN	0.128	0.469	0.979	100.2
	PVAR	0.128	0.468	0.978	101.5
	PNEW	0.129	0.469	0.979	119.0

Table 10: One Dimensional Test Results for the Waterfalls

R, dim	Alg.	Minimal (T.D.)	Mean (T.D.)	Maximal (T.D.)	Mean Oracle Calls
0.1, 10	CProX	0	7.066	9.538	18.3
	CPrFR	0	7.054	9.511	100.0
	N2FC1	0.068	0.964	2.431	94.2
	RGS	8.429	9.154	9.445	108.8
	PBUN	1.127	2.954	6.908	101.1
	PVAR	1.100	3.752	11.171	101.8
	PNEW	0.623	3.735	10.248	266.4
0.5, 10	CProX	1.417	4.114	7.010	4.9
	CPrFR	0	3.394	8.489	100.0
	N2FC1	0.163	0.811	4.029	90.0
	RGS	1.433	3.188	4.887	100.9
	PBUN	0.946	1.963	4.909	100.4
	PVAR	0.763	3.514	7.755	101.3
	PNEW	0.668	2.874	9.474	272.7
2.5, 10	CProX	0.265	0.620	1.093	2.0
	CPrFR	0.034	0.477	1.249	100.0
	N2FC1	0.199	0.673	1.672	94.2
	RGS	0.094	0.436	0.932	99.7
	PBUN	0.433	0.979	2.625	100.2
	PVAR	0.529	1.452	3.922	101.5
	PNEW	0.434	1.031	3.385	289.9

Table 11: Dimension Ten Test Results for the Waterfalls

R, dim	Alg.	Minimal (T.D.)	Mean (T.D.)	Maximal (T.D.)	Mean Oracle Calls
0.1, 100	C _{PROX}	93.142	96.281	97.252	100.0
	C _{PrFR}	93.142	96.281	97.252	100.0
	N2FC1	0.234	8.410	22.832	88.8
	RGS	1.975	1.984	1.993	103.0
	PBUN	7.782	15.416	49.660	100.3
	PVAR	9.121	18.009	40.524	102.3
	PNEW	9.651	32.161	60.514	1944.5
0.5, 100	C _{PROX}	26.866	44.847	64.529	6.7
	C _{PrFR}	0	11.635	86.820	100.0
	N2FC1	0.130	4.796	10.799	92.2
	RGS	1.775	1.784	1.793	103.0
	PBUN	6.651	12.464	43.265	101.1
	PVAR	8.769	21.543	67.492	102.3
	PNEW	7.934	18.163	59.066	1629.5
2.5, 100	C _{PROX}	5.033	6.655	8.178	2.0
	C _{PrFR}	0.739	7.252	9.356	100.0
	N2FC1	0.122	0.585	1.009	75.4
	RGS	0.775	0.784	0.793	103.0
	PBUN	3.482	4.391	5.194	100.8
	PVAR	3.565	8.641	22.465	101.6
	PNEW	3.345	11.637	25.088	2269.0

Table 12: Dimension One Hundred Test Results for the Waterfalls